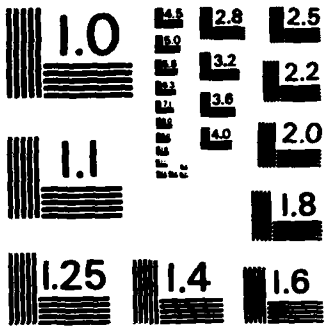
AD-A160 289 SPADE (SERIES PARALLEL DIRECTED ACYCLIC GRAPH EVALUATOR): A TOOL FOR PERF..(U) DUKE UNIV DURHAM NC DEPT OF COMPUTER SCIENCE R A SAHNER ET AL. JUL 85 1/1

UNCLASSIFIED CS-1984-15 AFOSR-TR-85-0745 AFOSR-84-0132 F/G 9/2 NL

END
FILMED
DTIC


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

REPORT DOCUMENTATION PAGE

AD-A160 289

1b RESTRICTIVE MARKINGS

3 DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4 PERFORMING ORGANIZATION REPORT NUMBER(S)

5 MONITORING ORGANIZATION REPORT NUMBER(S)
AFOSR-TR- 85-0745

6a NAME OF PERFORMING ORGANIZATION
Duke University

6b OFFICE SYMBOL *(If applicable)*

7a NAME OF MONITORING ORGANIZATION
Air Force Office of Scientific Research

6c ADDRESS *(City, State and ZIP Code)*
202 North Building
Durham, NC 27706

7b ADDRESS *(City, State and ZIP Code)*
Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332

8a NAME OF FUNDING/SPONSORING ORGANIZATION
AFOSR

8b. OFFICE SYMBOL *(If applicable)*
NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
AFOSR-84-0132

8c. ADDRESS *(City, State and ZIP Code)*
Bolling AFB DC 20332

10 SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	K3	

11. TITLE *(Include Security Classification)* SPADE: A Tool For Performance and Reliability Evaluation

12. PERSONAL AUTHOR(S)
Robin A. Sahner and Kishor S. Trivedi

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
Reprint	FROM ______ TO ______	July 1985	29

16. SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB. GR.
XXXXXXXXXXXXX		

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
directed acyclic graphs, semi-symbolic form

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

A model for the stochastic analysis of directed acyclic graphs is developed. These graphs represent event-precedence networks where the distribution function associated with an event is assumed to be a variant of the phase-type distribution. Events may occur sequentially, probabilistically, or concurrently. The distribution function of the graph execution time is computed in a semi-symbolic form. Applications of the model for the evaluation of concurrent program execution time and to the reliability analysis of fault-tolerant systems are discussed. [illegible]

[illegible]

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Brian W. Woodruff, Maj, USAF

22b TELEPHONE NUMBER *(Include Area Code)*
(202) 767-5027

22c. OFFICE SYMBOL
NM

AFOSR-TR- 85-0745

CS-1984-15
Revised Version

SPADE: A Tool For Performance and Reliability Evaluation

Robin A. Sahner and Kishor S. Trivedi

Department of Computer Science
Duke University

85 10 11 151

SPADE: A Tool For Performance and Reliability Evaluation

Robin A. Sahner and Kishor S. Trivedi

Department of Computer Science
Duke University
Durham, N. C. 27706

This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR-84-0132, by the Army Research Office under contract DAAG29-84-0045 and by the National Science Foundation under grant MCS-830200.

Abstract

A model for the stochastic analysis of directed acyclic graphs is developed. These graphs represent event-precedence networks where the distribution function associated with an event is assumed to be a variant of the phase-type distribution. Events may occur sequentially, probabilistically, or concurrently. The distribution function of the graph execution time is computed in a semi-symbolic form. Applications of the model for the evaluation of concurrent program execution time and to the reliability analysis of fault-tolerant systems are discussed.

1. INTRODUCTION

Many interesting problems in the design and analysis of multiple and distributed processing systems need to be solved in order to provide their designers and users with insights and tools for system evaluation. Most current work on performance and reliability analysis of parallel and distributed systems by means of analytic models may be classified as either *program-centered* (transaction-centered) or *resource-centered* analysis.

In resource-centered analysis, system resources are modeled in great detail while a relatively simple model of transaction behavior is assumed. Markov and semi-Markov models have been used for reliability and availability analysis [1,2]. Product-form queueing networks have been used to analyze the performance of computer systems [1,3], communication networks, and computer-communication networks [4]. The existence of a product-form solution implies a relatively efficient numerical procedure to obtain the solution. However, real system behavior rarely satisfies the necessary assumptions of a product-form network. For example, transactions with internal concurrency will violate product-form [5,6,7]. Many authors have studied approximate and exact solution techniques for solving queueing models where programs are allowed to overlap their computation with their own input-output operations [6,7].

In program-centered analysis, a relatively simple model of system resources is assumed while the characteristics of transactions are modeled in great detail. Behavior of program execution in the face of system failure/repair [8] and software failure [9] has been modeled. If we consider performance analysis in the absence of failures and assume that transactions possess internal concurrency, then transactions can be represented by precedence graphs [22]. Deterministic analysis of task precedence graphs and the scheduling of these graphs is known to be important [22]. Adding randomness to such graphs, we obtain stochastic activity networks [10,11]. At least three approaches to the analysis of such networks can be identified:

Markov chain techniques, stochastic Petri net techniques, and path analysis.

The first approach is to express the activity in the form of a continuous-time Markov chain [12,13]. This approach restricts the node times to be exponentially distributed, and also quickly leads to an explosion in the state-space of the Markov chain.

The second approach uses Petri net models. Ramamoorthy and Ho use this approach in the case that node times are deterministic [14]. Molloy considers exponentially distributed node times and converts the Petri net into a Markov chain for analysis [15]. We have allowed the node times to be generally distributed in our Extended Stochastic Petri Net (ESPN) Model [16]. Whenever possible, the ESPN is automatically converted into a Markov chain or a semi-Markov process. If neither of these approaches succeeds then the ESPN is evaluated using Monte-Carlo simulation. The first two approaches for the solution of an ESPN can lead to large state spaces, while the third approach can be time consuming due to the inherent speed limitations of a simulation model.

The approach developed in this paper falls into the path analysis category. The path analysis technique first computes the distribution of the time to traverse each path. For complex graphs the number of paths can be rather large, making the technique computationally expensive. In the general case, overlapping paths exist and hence one can only obtain an approximation (or bounds) for the overall execution time [11].

If the shape of the graph is restricted to series-parallel, the overall execution time can be obtained exactly. This is the approach taken by Robinson [17] and Kleinoder [18] in using directed graphs for the performance analysis of concurrent programs. We also do this, but our model allows for multiple paths to be interpreted in a variety of manners, not just as concurrent program execution. Therefore, our graphs can be used to model reliability as well as program execution. Kleinoder's approach differs from ours in that he performs numerical convolutions and other such operations on empirical distributions. Thus his approach avoids any distributional assumptions. However, our approach yields results in semi-symbolic form and is faster.

In this paper we consider the analysis of node-activity networks that are series-parallel graphs. With each node in the graph is associated a distribution that has exponential polynomial form. This form is quite general, and includes Neuts' phase-type distributions. The division of a graph into parallel subgraphs can be interpreted as either probabilistic or deterministic. In the deterministic case, the time needed to traverse all of the subgraphs may be either the maximum or minimum of the time needed to traverse the individual subgraphs. The distribution function of the total time to traverse the graph is studied. A program called SPADE (*Series-PArallel Directed* acyclic graph *Evaluator*) has been written to compute this distribution. Applications of the use of our model include performance analysis of concurrent programs and reliability analysis of non-repairable fault-tolerant systems.

In section 2, we discuss the model. Section 3 describes the analysis of the model and the SPADE program. In section 4 we give examples illustrating the use of our approach. The examples chosen are very simple, for the purpose of exposition. More complex problems can be and have been solved by the model.

2. THE SPADE MODEL

The SPADE model consists of a series-parallel acyclic directed graph with the nodes representing events and the edges representing a precedence relation between the events. Such graphs are useful for modeling many different kinds of activities. The two main applications are performance and reliability analysis. In this section we describe the SPADE model. First we give a performance model example and a reliability model example, to illustrate how directed graphs are used as models. Then we define series-parallel graphs and describe how they are interpreted by SPADE.

2.1. Graph Model Examples

As an example of a performance model, we consider the process communication graph from Kung's thesis [13], shown in figure 1. There are four tasks to be executed. Tasks 1 and 2 are executed on one processor and tasks 3 and 4 on another processor. Tasks 2 and 3 require results from task 1 and task 4 requires results from tasks 2 and 3. Once task 1 has completed, tasks 2 and 3 may be executed at the same time. When they are both finished, task 4 may be executed. Because tasks 1 and 3 and tasks 2 and 4 are assigned to different processors, data must be communicated between processors. The communication time between tasks 1 and 3 and tasks 2 and 4 is modeled by the nodes S_{13} and S_{24}. The execution time for the a task is exponentially distributed with parameter λ. The execution time for a data communication node is also exponentially distributed with parameter $s * \lambda$.

If the model is evaluated using the standard Markov approach, the corresponding Markov chain will have ten states. Subsequently, the Markov chain would be solved by any number of known techniques. SPADE avoids the state space expansion by analyzing the model directly. To further contrast the two approaches, the SPADE approach allows a much more general distribution type than exponential, but puts restrictions on types of graphs. Markov models allow cycles and general graphs but suffer from large state spaces, particularly if we wish to model non-exponential behavior.

To see how a directed graph can be used to model reliability, consider the system of components pictured in figure 2a. The system consists of three parallel subsystems, and functions if any one of the subsystems is working. Each subsystem is composed of a series of components, all of which must function in order for the subsystem to work.

Previous approaches to solving such reliability problems include combinatorial analysis and Markov chains[1]. Combinatorial analysis works well for "pure" series-parallel systems, but if we allow standby redundancy as well the combinatorial approach quickly becomes intractable. The Markov chain approach suffers from the curse of dimensionality; for an n-component system, the Markov chain contains 2^n states. The Markov chain approach also makes restrictive assumptions on distributions. The SPADE approach appears to overcome both of these problems. It should be mentioned, however, that the Markov chain approach does not impose the structural restrictions and independence assumptions of our approach.

To analyze the model using SPADE, the model would be transformed to the graph shown in figure 2b, which represents the time to failure of the system. For a series of components, we must take the minimum of the probability functions of the individual components (the series fails as soon as one component fails). For subsystems in parallel, we must take the maximum of the probability functions (the system fails only when all parallel subsystems fail).

2.2. Series-Parallel Graphs

The graphs in the two examples in the previous section are simple examples of the class of series-parallel graphs. There are several nearly equivalent definitions for the term "series-parallel" [20][21][25]; we define the term as follows. A finite linear graph is defined to be a quadruple $G=(N,E,S,T)$ where

- o N is a finite set of elements called nodes
- o E is a subset of $N \times N$, called the set of edges
- o S is the subset of N containing those nodes that are not the second member of any edge in E (these are the entrance nodes).
- o T is the subset of N containing those nodes that are not the first member of any edge in E (these are the exit nodes).

Suppose $G_1=(N_1, E_1, S_1, T_1)$ and $G_2=(N_2, E_2, S_2, T_2)$ are nonintersecting graphs. A graph $G=(N, E, S, T)$ is the series connection of G_1 and G_2 if and only if

- o At least one of T_1 and S_2 contains exactly one node.
- o $N=N_1 \cup N_2$
- o $E=E_1 \cup E_2 \cup (T_1 \times S_2)$
- o $S=S_1$, $T=T_2$

A graph G is the parallel combination of G_1 and G_2 if and only if

- o $N=N_1 \cup N_2$
- o $E=E_1 \cup E_2$
- o $S=S_1 \cup S_2$, $T=T_1 \cup T_2$

The class of series-parallel graphs is the smallest class of graphs containing the unit graphs (graphs consisting of one node) and having the property that whenever G is the series or parallel connection of two graphs in the class, then G is in the class. A series-parallel graph is by definition acyclic and contains no redundant edges.

Figure 3 shows an example of a series-parallel graph and the sequence of series and parallel combinations from which the graph can be built. Figure 4 shows two graphs which are *not* series-parallel.

2.3. Graph Interpretation

Each node in a graph represents an event whose length is specified by a cumulative distribution function (CDF). Given a graph, SPADE will compute the distribution function for the time taken to "traverse" the entire graph. The definition of what it means to "traverse" a graph G is recursive.

If G consists of a single node, the traversal time is given by the CDF associated with that node. If G was formed by combining the subgraphs G_1 and G_2 in series, then in order to traverse G, we must first traverse G_1 and then traverse G_2. If G is the parallel combination of G_1 and G_2, we allow the parallelism to be interpreted in one of three ways.

probabilistic

Only one of the subgraphs is actually traversed. Each subgraph has associated with it the probability that it is chosen for traversal. G has been traversed when one of the subgraphs has been traversed.

maximum

The two subgraphs are traversed concurrently. Traversal of G is complete when traversal of both G_1 and G_2 is complete.

minimum

The two subgraphs are traversed concurrently. Traversal of G is complete when traversal of the first subgraph to to finish is complete.

Suppose we are modeling program execution. Graphs with only probabilistic parallelism will model flowcharts of loop-free sequential programs. If we only allow maximum concurrency then the graphs will correspond to the task precedence graphs considered in [22]. Minimum concurrent subgraphs will model the parallel execution of a non-deterministic algorithm [23] in which the verification of all guessed solutions is attempted concurrently, and the first guess to be verified provides a solution to the whole problem.

The graphs can also be used to model the lifetime of closed (non-repairable) fault-tolerant systems with permanent faults. Such systems are defined in [24], where they are analyzed by Markov chain techniques. A system consisting of a series combination of components is modeled by parallel graph nodes with type minimum; a parallel combination is modeled by parallel graph nodes with type maximum. We should note that our graphs allow more general distributions of subsystem or component lifetimes than those allowed by the Markov chain techniques used in [24].

3. GRAPH ANALYSIS

The analysis of a SPADE model has two phases. First, the graph is decomposed into a binary tree. Then the tree is used to obtain the CDF for the graph traversal time. In this section we describe the two phases.

3.1. Tree Decomposition

Any series-parallel graph can be decomposed into a binary tree, where the internal nodes of the tree are of type "series" or "parallel", and the leaves of the tree are the nodes of the graph. Valdes, Tarjan and Lawler [20] present an elegant algorithm for performing the decomposition; the algorithm presented here is less elegant but simpler.

Suppose we have a graph G. Let $S = \{x_1, x_2, \cdots, x_n\}$ be the set of entrance nodes in G. For any node x, define $A(x)$ to be the set consisting of x itself and all descendants of x. Define B to be the set $\bigcap_{x \in S} A(x)$. Thus B consists of those nodes in G which are commonly descended from every entrance node. We have the following lemma:

Lemma:

Suppose G is the series combination of G_1 and G_2, and G_1 is the parallel combination of two subgraphs. Then the graph G_2 must have a single entrance node, and that node is the first node in B.

Proof:

By definition, since G_1 and G_2 were combined in series, either G_1 has a single exit node or G_2 has a single entrance node. G_1 has multiple exit nodes, because it has parallel, hence disjoint subgraphs, each of which has at least one exit node. Therefore G_2 has a single entrance node y.

When G_1 and G_2 were combined, all of the exit nodes in G_1 were connected to y. Every entrance node in G_1 is either itself an exit node of G_1, or has some exit node as its descendent. Therefore y is descended from every entrance node of G_1, and hence y is in B. Furthermore, because G_1 has disjoint subgraphs, no node in G_1 is descended from every entrance node of G_1, hence no node in G_1 is in B. Therefore y is the first node in B.

Armed with this lemma, we proceed to describe a recursive decomposition algorithm for series parallel graphs. For any graph G, let T_G be the tree which represents the decomposition of G. There are four cases.

case 1: $G = \{x\}$

T_G consists of the single node x.

case 2: $|G| > 1, S = \{x\}$

T_G consists of a root node of type "series" with left subtree the single node x and right subtree the tree $T_{G-\{x\}}$.

case 3: $|G| > 1, \ |S| > 1, B = \emptyset$

Since G has more than one entrance node and the entrance nodes have no common descendent, G can be divided into disjoint parallel subgraphs. Let $G_1 = G \cap A(x_1)$ and $G_2 = G - A(x_1)$. Note that $A(x_1)$ need not be a subset of G, since G may be some subgraph which contains x_1 but not all of its descendents. For each of $x_2, \cdots, x_n$, if $G_1 \cap A(x_i) \neq \emptyset$ then set $G_1 = G_1 \cup A(x_i)$ and $G_2 = G_2 - A(x_i)$. Then T_G consists of a root node of type "parallel" with left subtree T_{G_1} and right subtree T_{G_2}.

case 4: $|G| > 1, \ |S| > 1, B \neq \emptyset$

Since B is not empty, we cannot have a parallel decomposition, and hence must have a series decomposition. Let G_1 and G_2 be two series subgraphs of G such that G_1 is minimal. G_1 cannot consist of a single node, since then we would have $|S| = 1$. Therefore, G_1 must be composed of subgraphs, and those subgraphs must be parallel, otherwise G_1 would not be minimal. Now we have the conditions of the lemma, and hence the smallest node in B is the single entrance node of G_2. Let y be that node. Then $G_1 = G - A(y)$ and $G_2 = G \cap A(y)$. T_G consists of a root node of type "series" with left subtree T_{G_1} and right subtree T_{G_2}.

Figure 5 illustrates the decomposition process by showing an example of cases 2, 3 and 4. Figure 6 shows a series-parallel graph and the complete binary tree associated with it. Note that for SPADE the parallel nodes are divided into three types: "maximum", "minimum", and "probabilistic".

The tree decomposition is not necessarily unique, but all possible decompositions of a graph are equivalent in the sense that the probability distributions as computed by all of the possible binary trees are the same. This is true because "maximum" and "minimum" are associative, and for the probabilistic nodes, multiplication is distributive over addition.

3.2. The CDF of a Tree

Given a series-parallel directed graph and a CDF for the traversal time of each node, one can calculate the CDF of the traversal time for the entire graph by using the graph's decomposition tree. For a tree node A, let F_A be the CDF of the time needed to traverse the tree rooted at A. The calculation for F_A depends on the type of node A.

If A is a leaf, then A represents an event node in the series-parallel graph. F_A is simply the CDF specified for that node. Now suppose A is not a leaf, and that its two subtrees are B and C. If A is a *series* node, then A represents the traversal of the subtree B followed by the subtree C. F_A is given by

$$1)\quad F_A(t) = \int_0^t F_B'(x) F_C(t-x)\,dx$$

If A is a *maximum* node, then A represents the maximum of the traversal times of subtrees B and C; hence

$$2)\quad F_A(t) = F_B(t) F_C(t)$$

If A is a *minimum* node, then A represents the minimum of the execution times of subtrees B and C; therefore

3) $F_A(t) = F_B(t) + F_C(t) - F_B(t)F_C(t)$

Finally, if A is *probabilistic*, then only one subtree of A will be executed. Suppose the probability that subtree B is executed is p_B and the probability that subtree C is executed is p_C. Then F_A is given by

4) $F_A(t) = p_B F_B(t) + p_C F_C(t)$

It would be possible, given any series-parallel graph, to compute numerically the value of the CDF of the entire associated tree given any value of t. If the type of the CDF's is restricted to be of exponential polynomial form and the parameters are given, then the overall CDF will also be an exponential polynomial, and can be computed symbolically in terms of t. An exponential polynomial is defined to be an expression of the form

$$\sum_i a_i t^{k_i} e^{b_i t}$$

Restricting distributions to this form is a rather weak restriction, since it includes exponential, hyperexponential, Erlang and mixtures of Erlang distributions, in addition to Neuts' phase-type distribution.

Exponential polynomials are closed under the operations of addition, subtraction, multiplication, differentiation and integration. Because exponential polynomials are closed under these operations, a series-parallel graph whose nodes have exponential polynomials for CDF's will have an overall CDF which is also an exponential polynomial.

3.3. The SPADE Program

The process of finding the overall CDF of a series-parallel graph whose nodes have CDF's which are exponential polynomials can be automated. The program SPADE accepts a specification of such a graph and produces the overall CDF, mean and variance, and computes the overall CDF for a specified range of values.

In order to specify a model for SPADE, a user must supply, either interactively or in a file, four kinds of information: the graph edges, the parallelism types for parallel subgraphs, the distribution for each graph node, and a set of intervals over which to evaluate the overall graph CDF.

Because our definition of series-parallel requires subgraphs combined in series to begin or end with a single node, every group of parallel subgraphs is preceded by some single node, except when the overall graph has multiple entrance nodes. In that case, SPADE supplies a dummy entrance node. Thus, to specify the type of parallelism for parallel subgraphs, the user assigns an "exit type" to each node that has more than one immediate successor. The three choices for exit type are maximum, minimum or probabilistic. If the exit type is probabilistic, the user must specify the probabilities.

Every node in the graph must be assigned a cumulative distribution function. Each CDF must have exponential polynomial form. In general, for each CDF the user must provide the three parameters a_i, k_i and b_i for each term in the exponential polynomial. For the user's convenience, SPADE provides an abbreviation for the exponential distribution, so that the user need only supply the parameter (inverse of the mean) of the distribution.

SPADE also allows a node to have distribution "zero", meaning that the traversal time for the node is zero. These "zero" nodes are useful for representing synchronization, and to form graphs which would otherwise not adhere to series-parallel form. The examples in section 4 include instances where zero nodes are used.

Once all information about the graph has been obtained, the program forms a binary tree corresponding to the series-parallel graph. The graph decomposition results in a preorder representation of the tree. The program then backs up the preorder representation (traversing the tree in "reverse preorder"), computing the CDF of each subtree. Once the overall CDF is known, SPADE evaluates the CDF over intervals specified by the user.

4. APPLICATIONS

This section presents examples of applications of the task graphs defined in section 2. The applications are of two kinds: analysis of concurrent task execution and reliability analysis.

4.1. Concurrent Program Execution Time

Example 1. Consider a sequential program which consists of two phases. The outcome of the first phase determines which of two alternative tasks is executed as the second phase. Thus, the node representing the first phase has a probabilistic exit. This program is illustrated in figure 7a.

The execution time of task 1 (the first phase) is exponentially distributed with parameter $\lambda_1 = 0.6$. The execution time of task 2 is 2-stage Erlang distributed with parameter $\lambda_2 = 1$ (the CDF is $1-e^{-t}-te^{-t}$) and the execution time of task 3 is exponentially distributed with parameter $\lambda_3 = 0.8$. The probability that task 2 will be executed is $p = 0.9$ while the corresponding probability for task 3 is $1 - p$.

A specification of the model is shown in figure 7b. The edges of the graph are specified by the two node pairs (1,2) and (1,3). The exit type for node 1 is specified as probabilistic, meaning that the two parallel subgraphs {2} and {3} are probabilistic. The distributions for nodes 1 and 3 are given as exponential, with appropriate parameters. The distribution for node 2 is given by specifying three exponential polynomial terms. SPADE is asked to compute the overall CDF for values of t between 2 and 10, at intervals of 2.

The results given by SPADE are shown in figure 7c.

Example 2. Next we consider the producer-consumer problem. This problem is also explored in [19], in the context of real-time execution analysis. The producer process produces messages according to an exponential distribution with rate μ. The consumer process consumes the messages according to an exponential distribution with rate λ. We are interested in how long it takes for n messages to be produced and consumed.

The process of the production and consumption of two messages is shown by the graph in figure 8a. The nodes P1 and P2 represent the production of the first and second messages; C1 and C2 represent the consumption of the messages. Obviously, each message cannot be consumed until it is produced, but it is possible for message one to be consumed while message two is being produced.

With $\mu = .5$ and $\lambda = .3$, we have the input file and results in figures 8b and 8c, respectively.

Example 3. We evaluate one iteration of the program with CPU-I/O overlap considered by Towsley, Chandy and Browne [7] and shown in figure 9a. In each iteration of the program, there are two stages. The first stage is always a CPU burst. The second stage consists of either pure input/output, or input/output that may be overlapped with a second CPU burst. The probability that the second stage consists of CPU-I/O overlap is given by p. Note the use of a "zero" node. This allows us to have one branch of the CPU1 node lead to a single node, while the other branch leads to a group of nodes to be executed in parallel.

Assuming μ_1=0.125, μ_2=0.0376, λ=0.0217, and p =0.6, the results from SPADE are given in figure 9b. Note that in SPADE the distributions could have been more general.

Example 4. In this example we consider the process communication graph [13] discussed in section 2.1 and pictured in figure 1. Assuming that λ=0.125 and s =10, the distribution function and the mean and variance of the overall execution time computed by SPADE are in figure 10.

4.2. Reliability Analysis

We now consider a sequence of examples of the use of SPADE for reliability analysis.

Example 5. Consider the series-parallel system discussed in section 2.1 and illustrated in figures 2a and 2b. The system can be specified for SPADE as in figure 11a. Note the use of "zero" nodes. The results are given in figure 11b.

Example 6. We consider a triple modular redundant (TMR) system. Such a system consists of three components, any two of which must be functioning in order for the system to work. A task graph for the lifetime of this system can be derived based on example 3.24 of [1] and is shown in figure 12a. Nodes 2, 3, and 4 combined in "minimum" parallelism represent the time to failure of the first component to fail. Nodes 6 and 7 represent the time to failure of the second component to fail. In this case the distributions must be exponential, since we rely on the memoryless property of the exponential distribution when modeling the system in this manner. The solution from SPADE is given in figure 12b.

It is possible to simplify the task graph for this example as shown in figure 12c (based on example 3.26 of [1]). The results are, of course, the same.

5. CONCLUSION

A model for the performance and reliability analysis using directed acyclic graphs is discussed here. The attractiveness of the model lies in an efficient procedure for its evaluation, potentially wide class of applications, and a powerful class of allowed distributions. The restriction of the model involves structural restrictions on the kinds of models allowed. Structural restrictions include an inability to allow cycles and the restriction to series-parallel graphs. Work is currently under way to extend the model so as to remove some of the restrictions.

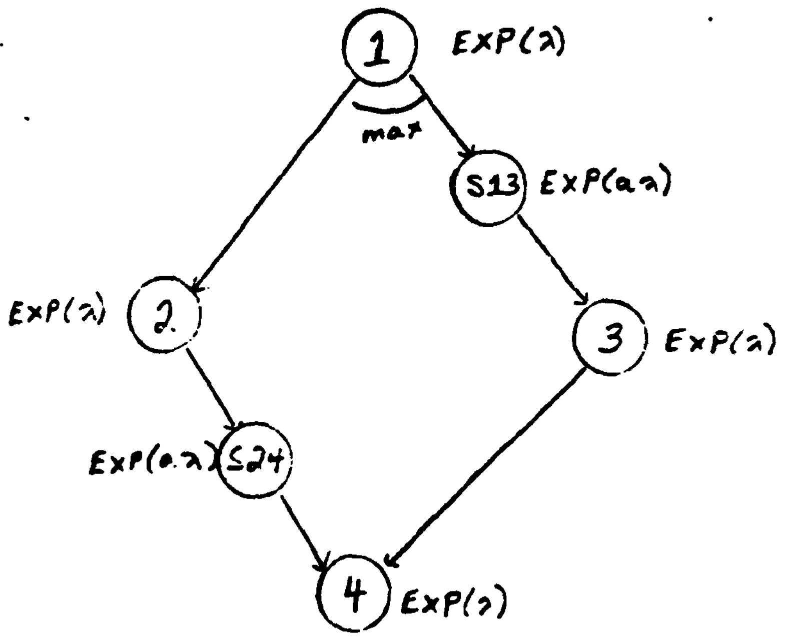
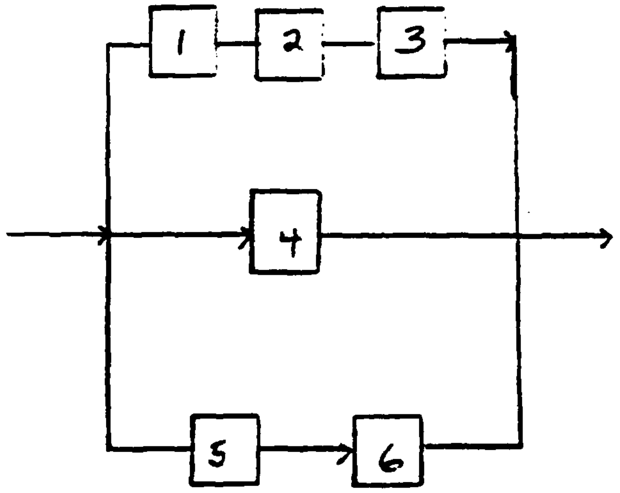


Figure 1. Process Communication Graph



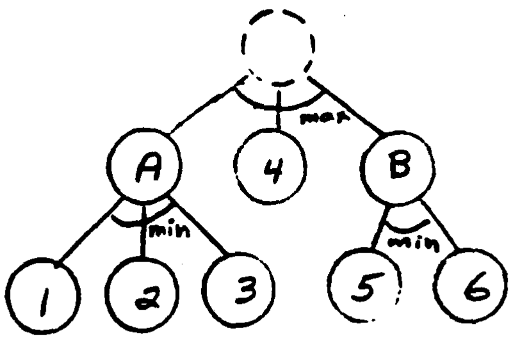


Figure 2a. Series-Parallel Components

Figure 2b. Graph Representation

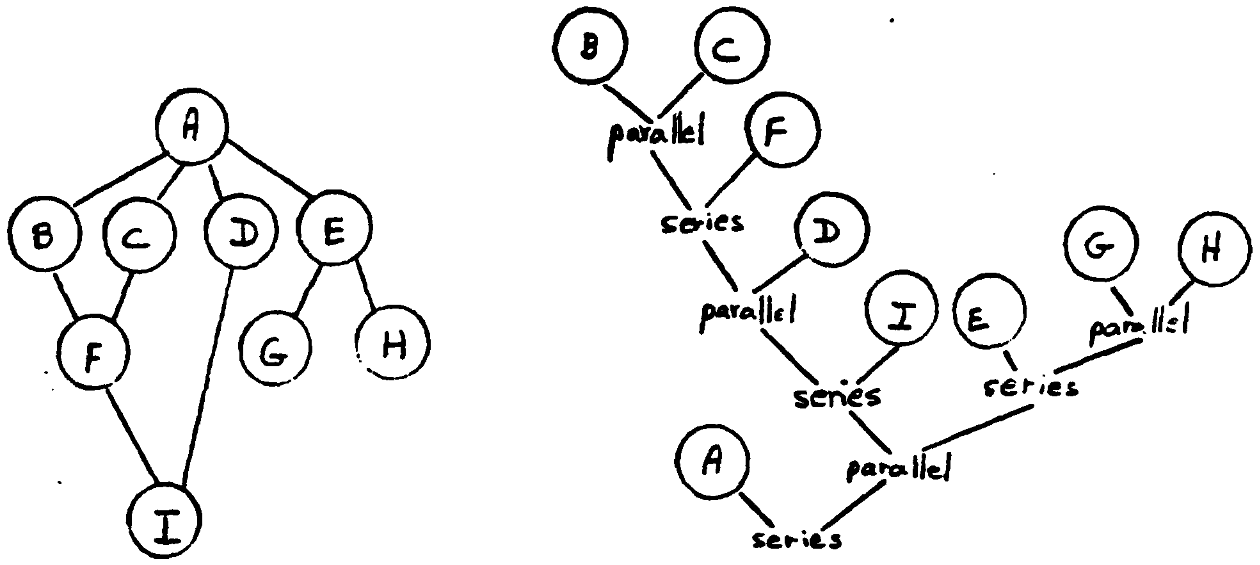


Figure 3. A Series-Parallel Graph

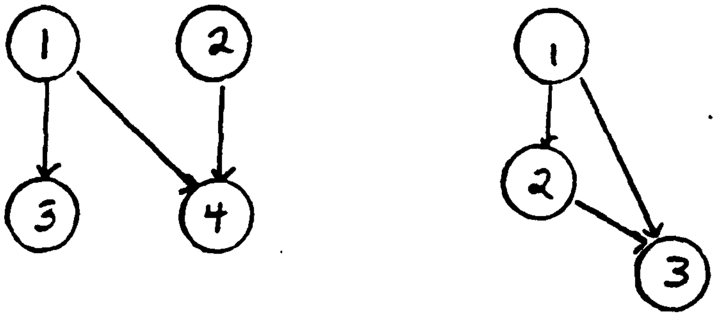
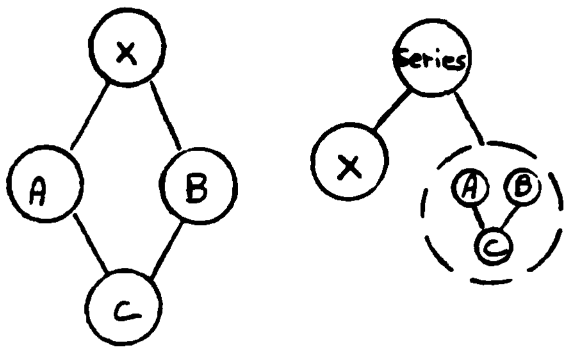
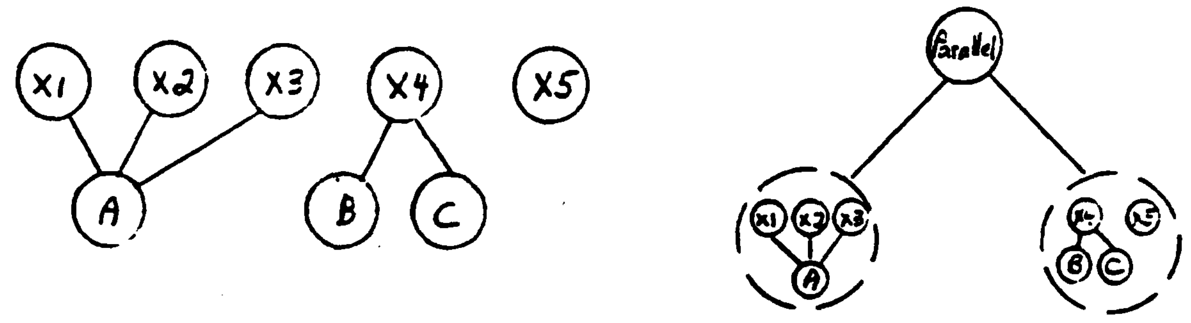


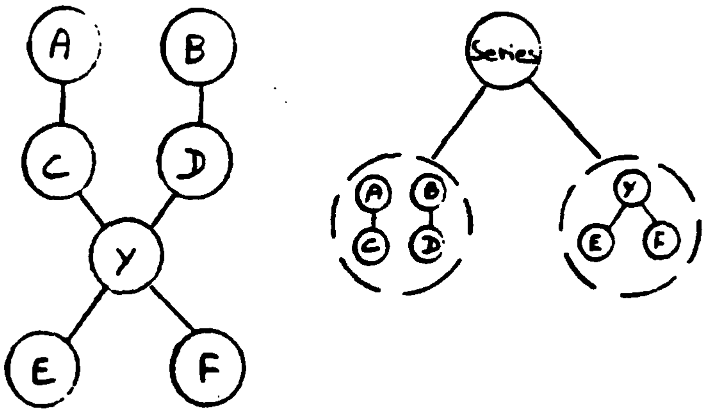
Figure 4. Two Graphs that are not Series-Parallel



Case 2



Case 3



Case 4

Figure 5. Decomposition Algorithm

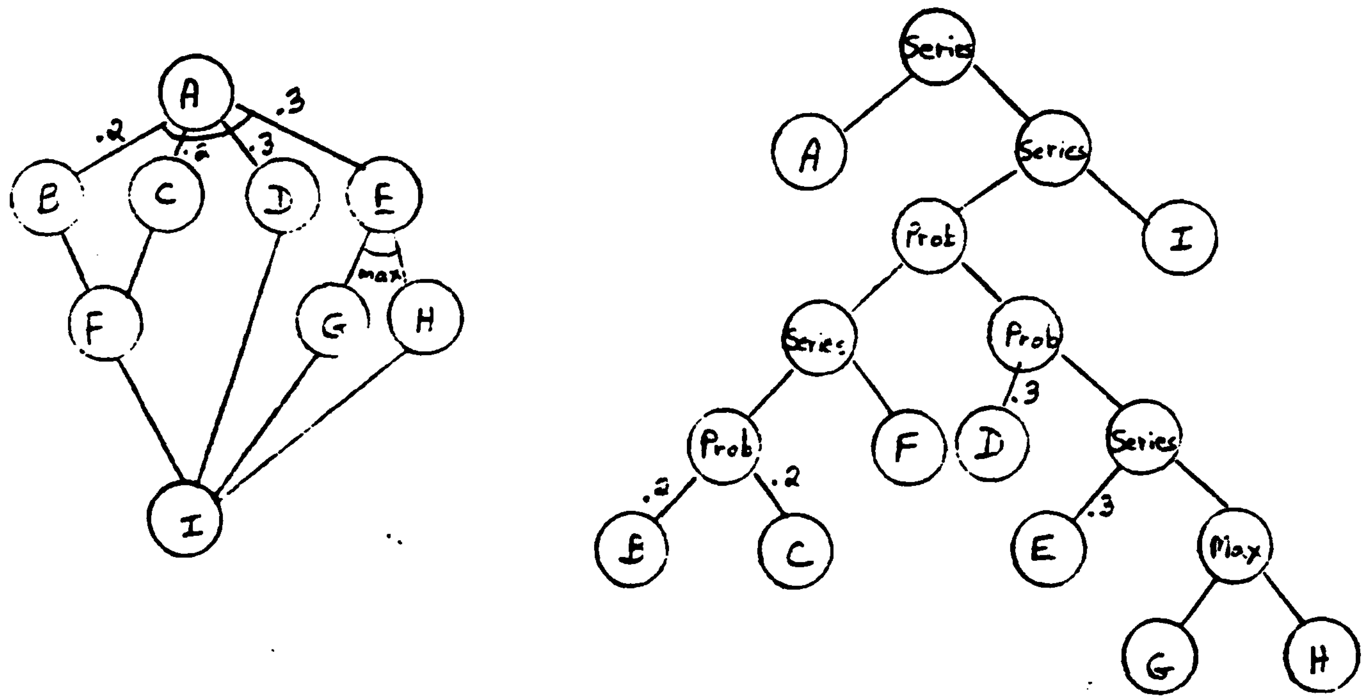


Figure 6. Binary Tree Decomposition

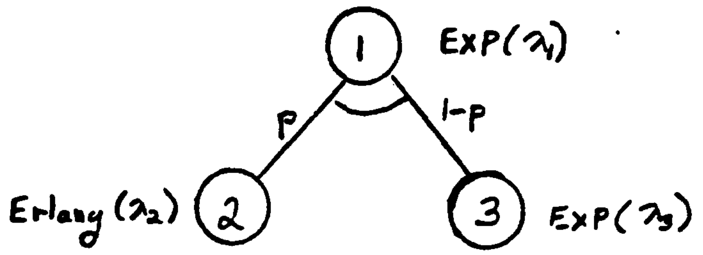


Figure 7a. Program with Probabilistic Tasks

```
GRAPH
ARC       1       2
ARC       1       3
END
EXIT      1     PROB
PROB      1       2 .9
DIST      1      EXP .6
DIST      2      GEN
EXPO        -1,   1, -1
EXPO         1,   0,  0
EXPO        -1,   0, -1
ENDGEN
DIST      3      EXP .8
END
EVAL   2 10 2
```

Figure 7b. Input for Example 1

```
CDF:
      1.3500 t( 1) exp(   -1.0000t)
+     1.0000 t( 0) exp(    0.0000t)
+    -6.0250 t( 0) exp(   -0.6000t)
+     0.3000 t( 0) exp(   -0.8000t)
+     4.7250 t( 0) exp(   -1.0000t)

mean: 3.5917
variance: 4.7847

    t       F(t)
   2.0     0.2507
   4.0     0.6511
   6.0     0.8696
   8.0     0.9561
  10.0     0.9860
```

Figure 7c. Results for Example 1

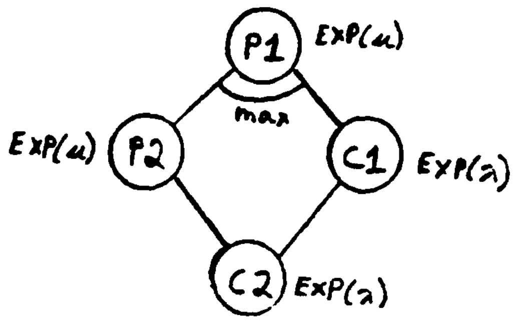


Figure 8a. Consumer-Producer Problem

```
COMM   consumer-producer problem
COMM   with 2 messages

GRAPH
ARC P1 P2
ARC P1 C1
ARC P2 C2
ARC C1 C2
END

EXIT P1 MAX
DIST C1 EXP .3
DIST C2 EXP .3
DIST P1 EXP .5
DIST P2 EXP .5
END

EVAL 4 20 5
END
```

Figure 8b. Input for Example 2

```
CDF:
 -0.7500 t( 1) exp(   -0.3000t)
+   0.7500 t( 1) exp(   -0.5000t)
+   1.0000 t( 0) exp(    0.0000t)
+  -1.0000 t( 0) exp(   -0.3000t)
+  -1.0000 t( 0) exp(   -0.5000t)
+   1.0000 t( 0) exp(   -0.8000t)

mean:  9.4167
variance: 25.5347

t        F(t)
4.0      0.1067
9.0      0.5438
14.0      0.8362
19.0      0.9500
```

Figure 8c. Results for Example 2

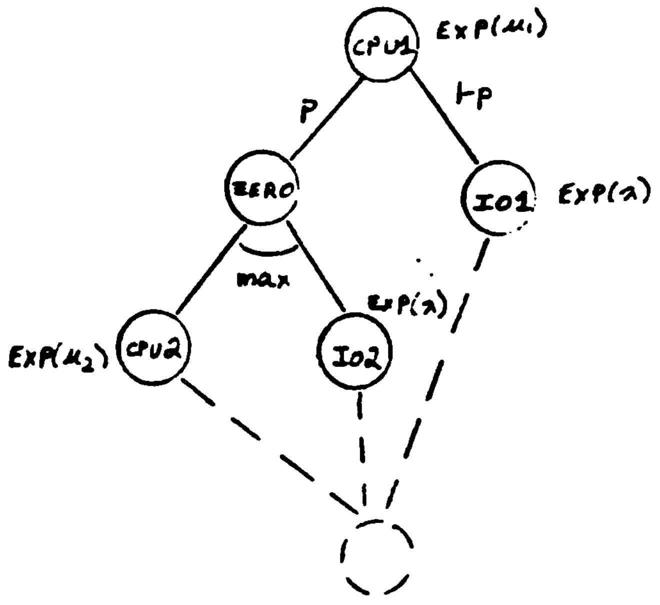


Figure 9a. CPU-I/O Overlap

```
CDF:
      1.0000 t( 0) exp(    0.0000t)
+    -1.2101 t( 0) exp(   -0.0217t)
+    -0.8581 t( 0) exp(   -0.0376t)
+     1.1416 t( 0) exp(   -0.0593t)
+    -0.0734 t( 0) exp(   -0.1250t)

mean: 59.9224
variance: 2122.8968
```

Figure 9b. Results for Example 3

```
CDF:
        -0.0174 t( 2) exp(   -0.1250t)
  +      0.0294 t( 1) exp(   -0.1250t)
  +      1.0000 t( 0) exp(    0.0000t)
  +     -2.2349 t( 0) exp(   -0.1250t)
  +      1.2346 t( 0) exp(   -0.2500t)
  +      0.0027 t( 0) exp(   -1.2500t)
  +     -0.0025 t( 0) exp(   -1.3750t)
  +      0.0000 t( 0) exp(   -2.5000t)

mean: 28.8364
variance: 208.1388

    t       F(t)
  10.0     0.0479
  30.0     0.6014
  50.0     0.9147
  70.0     0.9865
  90.0     0.9982
```

Figure 10. Results for Example 4

```
COMM        series-parallel system

GRAPH
ARC A 1
ARC A 2
ARC A 3
ARC 4
ARC B 5
ARC B 6
END

EXIT ENTRANCE MAX
EXIT A        MIN
EXIT B        MIN
DIST A        ZERO
DIST B        ZERO
DIST 1        EXP    .4
DIST 2        EXP    .6
DIST 3        EXP    .4
DIST 4        EXP    .8
DIST 5        EXP    .9
DIST 6        EXP    .3
END

EVAL 2 10 2
END
```

Figure 11a. Input for Example 5

```
CDF:
1.0000 t( 0) exp(          0.0000t)
+    -1.0000 t( 0) exp(   -0.8000t)
+    -1.0000 t( 0) exp(   -1.2000t)
+    -1.0000 t( 0) exp(   -1.4000t)
+     1.0000 t( 0) exp(   -2.0000t)
+     1.0000 t( 0) exp(   -2.2000t)
+     1.0000 t( 0) exp(   -2.6000t)
+    -1.0000 t( 0) exp(   -3.4000t)

mean: 1.7526
  variance: 1.4267

  t       F(t)
 2.0     0.6816
 4.0     0.9478
 6.0     0.9908
 8.0     0.9983
10.0     0.9997
```

Figure 11b. Results for Example 5

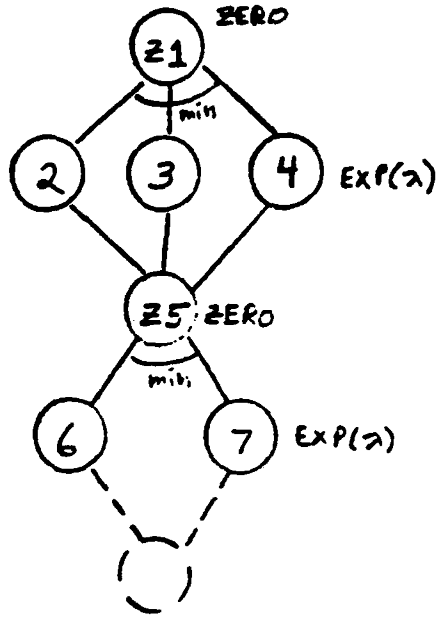


Figure 12a. TMR System

```
CDF:
        1.0000 t( 0) exp(     0.0000t)
  +    -3.0000 t( 0) exp(    -0.0002t)
  +     2.0000 t( 0) exp(    -0.0003t)

mean: 8333.3333
variance: 36111111.1111

     t          F(t)
  2000.0       0.0867
  4000.0       0.2544
  6000.0       0.4270
  8000.0       0.5757
 10000.0       0.6936
```

Figure 12b. SPADE results

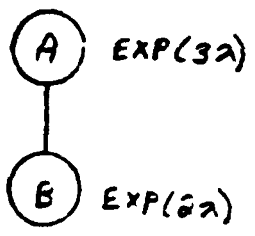


Figure 12c. Equivalent TMR System

REFERENCES

[1] Trivedi, K.S., ***Probability and Statistics with Reliability, Queueing and Computer Science Applications,*** Prentice-Hall, Englewood Cliffs, N.J., 1982.

[2] Trivedi, K., Dugan Bechta J, Geist R., and Smotherman ,M., "Modeling Imperfect Coverage in Faul-Tolerant Systems," Proc. of the Fourteenth Int. Conf. on Fault-Tolerant Computing (FTCS - 14), Orlando, FL., June 1984,pp. 77-82.

[3] Chandy, K.M., Howard, J.H., and Towsley, D.F., "Product Form and Local Balance in Queueing Networks," *JACM,* Vol. 24, pp. 250-263.

[4] Kleinrock, L., ***Queueing Systems, Vol. II: Computer Applications,*** John Wiley & Sons, 1976.

[5] Heidelberger, P. and Trivedi, K.S., "Queueing Network Models for Parallel Processing with Asynchronous Tasks," ***IEEE Trans. on Computers,*** November 1982.

[6] Heidelberger, P. and Trivedi, K.S., "Analytic Queueing Models for Programs with Internal Concurrency," ***IEEE Trans. on Computers,*** January 1983.

[7] Towsley, D.F., Browne, J.C. and Chandy, K.M., "Models for Parallel Processing within Programs," *CACM,* October 1978.

[8] Kulkarni, V., Nicola , V. and Trivedi, K., "On Modeling the Performance and Reliability of Multi-Mode Computer Systems," Proc. Int. Workshop on Modeling and Performance Evaluation of Parallel Systems, Grenoble , Dec. 1984, (to be published by North-Holland).

[9] Littlewood, B., "A Semi-Markov Model for Software Reliability with Failure Costs", Proceedings of the Symposium on Computer Software Engineering, New York, 1976, pp. 281-300.

[10] Fix, W. and Neumann, K., "Project Scheduling by Special GERT Networks," ***Computing*** 23 (1979), pp. 299-308.

[11] Gaul, W., "On Stochastic Analysis of Project Networks," in M.A.H. Dempter et al. (eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Co., 1982.

[12] Kulkarni, V. and Adlakha, W., "Markov and Markov-Regenerative PERT Networks", Tech. Report, Operations Research and Systems Analysis, Univ. of North Carolina at Chapel Hill, 1984.

[13] Kung, K.C.-Y., "Concurrency in Parallel Processing Systems", Ph.D. Dissertation, UCLA Computer Science Department, 1984.

[14] Ramamoorthy, C.V., and Ho, G.S., "Performance Analysis of Asynchronous Concurrent Systems Using Petri Nets", *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 5, pp. 440-449, September 1980.

[15] Molloy, M., "On the Integration of Delay and Throughput Measures in Distributed Processing Models", Ph.D. dissertation, Computer Science Department, UCLA, 1981 .

[16] Dugan, J.B., Trivedi, K.S., Geist, R.M. and Nicola, V.F., "Extended Stochastic Petri Nets: Analysis and Applications", accepted, PERFORMANCE '84, Paris, December 1984.

[17] Robinson, J.T., "Some Analysis Techniques for Asynchronous Multiprocessor Algorithms," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 1, January 1979.

[18] Kleinoder, W., "Evaluation of Task Structures for a Hierarchical Multiprocessor System", *Proc. Int. Conf. on Modeling Techniques and Tools for Performance Analysis*, Paris, France, May 1984.

[19] Haase, V. R., "Real-Time Behavior of Programs", *IEEE-TSE*, September 1981

[20] Valdes, J., Tarjan, R.E., and Lawler, E.L., "The Recognition of Series-Parallel Digraphs", Siam J. of Computing, Vol. 11 no 2 (1982) pp. 298-313.

[21] Abdel-Wahab, H.M. and Kameda, T., "Scheduling to Minimize Maximum Cumulative Cost Subject to Series-Parallel Precedence Constraints", Operations Research Vol 26, No. 1 pp. 141-158.

[22] Coffman, E.G., *Computer and Job/Shop Scheduling*, John Wiley & Sons, NY., 1976.

[23] Sedgewick, R., *Algorithms*, Addison-Wesley, Reading, Mass., 1983.

[24] Ng, Y.-W. and Avizienis, A., "A Model for Transient and Permanent Fault Recovery in Closed Fault-Tolerant Systems," *Proc. 1976 Int. Symp. on Fault-Tolerant Computing*, June 1976.

[25] Elgot, C.C. and Wright, J.B., "Series-Parallel Graphs and Lattices", *Duke Mathematics Journal*, vol. 26 (1959), pp. 325-338.